apple computer

Apple *III* Sophisticated Operating System (SOS)

Operating Systems

Simplicity in System Control

The Apple III Sophisticated Operating System (SOS) is a powerful software interface that helps you exploit Apple III's advanced system capabilities. With SOS, you don't need to worry about most internal system functions, such as which disk drive contains a file, which bank of memory is being used, or which slot a peripheral interface card occupies. These and other functions are all controlled automatically.

Designed for easy expandability as well as operational convenience, SOS enhances and ensures Apple III system flexibility. A simple-to-use SYSTEM CONFIGURATION PROGRAM (supplied on diskette) allows even inexperienced users to custom-configure SOS to meet their specific needs.

SOS provides a solid foundation for writing advanced applications on the Apple III. It features a hierarchical file system, device level interrupt capabilities, user level interrupt capability (events), a device-independent file system, and memory management capabilities. And, since all languages on the Apple III use SOS, they all share a common disk format. For instance, a Pascal application program can access a BASIC text file just as easily as it can access a Pascal text file.

Apple's Sophisticated Operating System for the Apple III gives you the efficiency, comprehensiveness, and expandability you've always wanted in an operating system, with a degree of convenience you may never have thought possible. SOS is supplied with all Apple III systems and software.

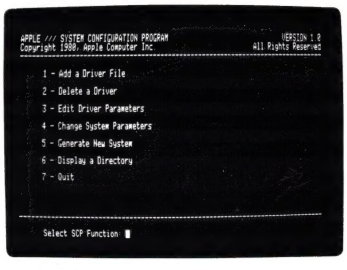
Benefits

The Apple *III* Sophisticated Operating System...

- *improves personal productivity, because it frees you from most system control responsibilities...*
- *simplifies programming, because it provides powerful, standard device and file interfaces for all languages and applications...*
- *maximizes system efficiency by automatically controlling the use and allocation of system resources...*
- *speeds up software development by reducing program size and complexity...*
- *ensures system adaptability, because it's designed for easy expansion and custom configuration.*

The Apple *III* Sophisticated Operating System— A Closer Look

SOS, which consists of five basic components, acts as a buffer between user programs and the Apple *III* hardware. Using these components, SOS controls Apple *III* system operation and resource allocation with a minimum of user involvement. As far as the user is concerned, handling the system's input/output (I/O) devices is as easy as handling its data files once SOS has been configured with the proper drivers.


APPLE /// SYSTEM CONFIGURATION PROGRAM
Copyright 1980, Apple Computer Inc.
VERSION 1.0
All Rights Reserved
1 - Add a Driver File
2 - Delete a Driver
3 - Edit Driver Parameters
4 - Change System Parameters
5 - Generate New System
6 - Display a Directory
7 - Quit
Select SCP Function:

Configuring SOS is a quick and easy process, even for the most inexperienced user. Simply follow the directions in the SYSTEM CONFIGURATION PROGRAM, one of the programs contained on the System Master diskette supplied with your Apple *III*.

Suppose, for example, that you've added an Apple *III* Silentype Printer to your system. To configure SOS so it will automatically control the printer, execute the SYSTEM CONFIGURATION PROGRAM and select the first option on the menu, ADD A DRIVER FILE. Then simply follow the directions displayed on the screen.

Once you've configured SOS to control your Silentype, you need never concern yourself again with its system operational requirements. SOS handles them for you—automatically—whenever you use the printer. Configuring SOS for other printers and peripherals is just as easy.

You don't have to be a system architect to get complete access to the Apple *III's* advanced capabilities. Apple's Sophisticated

Operating System helps you control and configure your Apple *III*, maximizing its benefits and power in every configuration and application.

Technical Specifications

The Structure of SOS

At the core of SOS is the "Kernel," a set of programs and subroutines that control the flow of information around and through SOS. The Kernel is divided into five main areas: the File Manager, the Device Manager, the Memory Manager, the Interrupt Manager, and the Utility Manager.

The File Manager controls the logical storage, transfer and routing of information within the Apple. All information is stored in units called files. The File Manager can create and destroy files, read their contents, write new information into them, change their names, or move them from place to place.

The Device Manager controls the physical storage and flow of information into and out of the Apple *III*. Together with its associated device drivers, it controls the operation of the console (the screen and keyboard), the serial port, the printer port, the disk drives, and all other peripheral devices connected to the Apple.

The Memory Manager in SOS allows programs and languages to use all the memory you have in your Apple quickly and efficiently. It keeps track of "banks" and "pages" of memory, and allocates their use to different programs and languages.

The Interrupt Manager works with the Device Manager to allow devices such as the keyboard or the serial port to interrupt the Apple in the middle of an operation. The Apple can then attend to the interrupting device, and resume its previous operation as if nothing had happened. The operation of the Interrupt Manager, like that of the Memory Manager, is normally invisible to you.

The Utility Manager lets programs access the built-in joystick interfaces. Most languages and applications that use the joysticks will have more convenient, higher-level commands to read the status of those devices. These high-level commands simply request the information from the Utility Manager through SOS.

Besides those components of SOS making up the Kernel, there are two other components. One is System Utilities, which provides high-level routines for performing more complicated system tasks with a minimum of user interaction. Another is the System Configuration Program, which allows custom configuration of SOS and the I/O device drivers, in order to match the user's specific needs.

File Management System Calls

CREATE (pathname,create_list,length)
DESTROY (pathname)
RENAME (pathname,new_pathname)
SET_FILE_INFO (pathname,file_list,length)
GET_FILE_INFO (pathname,file_list,length)
VOLUME (dev_name,vol_name,free_blocks)
SET_PREFIX (prefix_path)
GET_PREFIX (prefix_path,length)
OPEN (pathname,ref_num,open_list,length)
NEW_LINE (ref_num,is_newline,newline_char)
READ (ref_num,buf,bytes,bytes_read)
WRITE (ref_num,buf,bytes)
CLOSE (ref_num)
FLUSH (ref_num)
SET_MARK (ref_num,base,displacement)
GET_MARK (ref_num,mark)
SET_EOF (ref_num,base,displacement)
GET_EOF (ref_num,eof)

Device Management System Calls

D_CONTROL (dev_num,control_code,control_list)
D_STATUS (dev_num,status_code,status_list)
GET_DEV_NUM (dev_name,dev_num)
D_INFO (dev_num,dev_name,dev_list,length)

Utility Management System Calls

SET_FENCE (priority)
GET_FENCE (priority)
JOYSTICK (j_mode,j_value)

Memory Management System Calls

REQUEST_SEG (base,limit,seg_id,seg_num)
FIND_SEG (srch_mode,seg_id,pages,base,limit,seg_num)
CHANGE_SEG (seg_num,chg_mode,pages)
GET_SEG_INFO (seg_num,base,limit,pages,sed_id)
GET_SEG_NUM (bankpage,seg_num)
RELEASE_SEG (seg_num)

Apple Computer Inc. 10260 Bandley Drive Cupertino, California 95014 (408) 996-1010 TLX 171-576
©1981 Apple Computer Inc.
Printed in U.S.A. 11/81 250M Order No. A2F0037

apple computer

Apple Business BASIC

Apple *III's* Advanced BASIC for Business Applications

Languages and Utilities

Apple Business BASIC for the Apple III Computer System is one of the most powerful versions of BASIC ever developed for a microcomputer. Designed for those who want the flexibility, power, and ease of use of the popular BASIC language, Apple Business BASIC incorporates a number of innovative features which can be used to satisfy some of the most demanding business and scientific programming needs.

With Apple Business BASIC, programmers can address over 70K bytes of extended memory—the largest workspace available on any personal computer. In addition, Apple Business BASIC's special 64-bit, 18-digit data type handles the toughest accounting chores with "penny accuracy."

For producing reports, Apple Business BASIC offers some of the most comprehensive and flexibly formatted output capabilities found on any version of BASIC—even those used on large system mainframes. Apple Business BASIC also features: advanced file handling, with flexible file formats to match application needs; 16-bit binary integer, and 32-bit binary floating point data types; 64-character variable names (all characters significant) for documentation purposes; easy access to a wide variety of plug-in peripherals or auxiliary devices; and a trace mode for debugging.

If you prefer programming in BASIC, but require more power, flexibility, and precision than most BASICs can give you, turn to Apple Business BASIC for the Apple III—an advanced language for an advanced computer system. Apple Business BASIC is supplied with all Apple III system configurations.

Benefits

Apple Business BASIC...

- *makes it easier to generate reports, because of its flexible, extensive, formatted output capabilities, and versatile PRINT USING and IMAGE statements...*
- *increases system efficiency, because its large, user-available workspace (over 70K bytes) allows lengthy programs to be kept in memory...*
- *adds programming flexibility, because it lets you keep multiple (up to 10) files open simultaneously, with virtually no size constraints...*
- *allows you to display and calculate financial accounting data with "penny accuracy" through the use of a 64-bit, 18-digit data type and special functions...*
- *speeds up disk file storage, because its built-in facilities let programs read and write files more efficiently...*
- *simplifies system input/output (I/O) control, because just a few key words let you control a wide variety of peripherals, as well as machine language routines and graphics facilities...*
- *aids development and debugging of complex programs, because its optional ELSE statement and automatically indented listings encourage a structured approach to programming.*

Apple Business BASIC— A Closer Look

Apple Business BASIC is a general purpose, problem-solving language designed especially for business and scientific applications calling for: 1) an easy-to-use debugging environment; 2) fast, high-precision operations—such as accounting—on numbers with a range less than $\pm 10^{18}$; 3) a high degree of output formatting for printing reports; and 4) greater flexibility for file I/O operations. Apple Business BASIC's powerful features provide the programmer with a number of advanced capabilities and conveniences.

Extended User Program Memory Space

In an Apple *III* Computer System with 128K bytes of RAM, Apple Business BASIC presents you with a huge 70K-byte workspace—

more than is available with any other personal computer BASIC. This means you can write large programs more easily and run them more efficiently. Access time is reduced, because large files can be kept in memory—instead of on disk—and because many files can be opened for access simultaneously. In addition, large, high-resolution graphics areas can be used without fear of unduly restricting program size.

Flexible Output Formatting
Apple Business BASIC's versatile PRINT USING and IMAGE statements allow you to use a variety of format strings to prepare reports. Specifications are extremely flexible, and include string, literal, digit, scientific notation, and engineering notation categories.

Advanced File Handling
In Business BASIC, file handling is done through numbered file references defined within the program. You can make these file definitions perform chores simply by using generic terms. To print, for example, call out the generic term ".printer" within the program. Apple Business BASIC works with the Apple ///'s Sophisticated Operating System (SOS) to handle the details automatically, including which printer you're addressing (if your system has more than one), its I/O address, etc. For greater user convenience, disk files are referenced by directory, sub-directory, and file name, without regard to the storage device on which the file resides.

Long Variable Names
Apple Business BASIC allows you to use variable names (up to 64 characters in length) for documentation purposes, with each character significant. Furthermore, because spaces are used as delimiters, embedded BASIC keywords are allowed in variable names, permitting even more flexibility.

"Structured" BASIC Features
With Apple Business BASIC, you can add the ELSE statement to the usual IF...THEN statement provided by BASIC. And your ELSE clauses can themselves contain other IF...THEN...ELSE constructs. Apple Business BASIC also provides a LISTing feature that automatically indents the contents of FOR...NEXT loops. These "structured" BASIC features encourage programmers to use a more logical, structured approach to developing their programs, and greatly aid development and debugging of complex programs.

Powerful Interface to Assembly Language
Because Apple Business BASIC runs in the SOS environment, you don't have to worry about memory management, buffer allocation, or file handling. This freedom will save you a significant amount of programming time.

Additionally, Apple Business BASIC relates to SOS through a powerful Invoke/Perform interface mechanism. For example, if you have created assembly language routines using the disk assembler, you can specify the assembled routine by name in an Apple Business BASIC program. Once the routines are mentioned in the "INVOKE" statement, Apple Business BASIC works with SOS to find a residence in memory for the routines, and to establish—as entry points in the resultant linked module—any function or procedure names mentioned in the routines. All you have to do is PERFORM the previously invoked routine—specifying any variables to be passed—and Apple Business BASIC and SOS automatically handle all the operational details.

Apple Business BASIC makes life a lot easier for programmers by expanding system capabilities, reducing program development time, and adding greater flexibility to formatted output and file handling facilities.

Technical Specifications

Format
16-sector diskette

Variables
- 64 characters (max.), all significant
- Reserved variables:
  ERR, KBD, EOF, VPOS, ERRLIN, HPOS, FRE, PREFIX$

- Data Types
  16-bit binary integers (−32768 to +32767)
  64-bit binary integers (±9223372036854775807 or $2^{63} - 1$)
  32-bit binary floating point ($\pm 10^{38}$ with 6-digit precision)
  Character strings (0 to 255 characters, dynamic)
  String and numeric arrays (indexed starting with 0, no dimensional limits)

Operators

- General: +, −, *, /, , DIV, mod
  (Note: DIV and MOD are only for long integer operations)
- Binary logical operators:
  AND, OR, =, <, >, <>, ><, >=, <=, =<, =>
- Unary logical operator:
  NOT
- String operator:
  + (concatenation)

Statements

(Note: No statement or statement list may exceed 254 characters, including delimiters.)
LET (optional)
REM
GOTO
IF...GOTO
IF...THEN
IF...statementlist: ELSE statementlist
FOR ctrl variable = expression TO expression STEP expression
NEXT ctrl variable (,other ctrl variable)
GOSUB
RETURN
POP
ON expression GOTO
ON expression GOSUB
ON ERR
ON KBD
ON EOF #
OFF ERR
OFF KBD
OFF EOF #
RESUME

Utility Statements:

NEW	LOAD	STOP
CLEAR	SAVE	END
FRE	DELETE	CONT
PREFIX$	RUN	CHAIN prog name, line number

User-Defined Functions:
FN functionname (argument)
DEF FN functionname (argument) = expression

Debugging:
TRACE
NOTRACE

Cursor and Screen:

LIST	VPOS	HOME	NORMAL
DEL	HPOS	INVERSE	TEXT

String, Numeric and File Functions:

LEN	TEN	CONV	TAN	SQR
STR$	MID$	CONV%	ATN	EXP
VAL	LEFT$	CONV&	INT	LOG
CHR$	RIGHT$	CONV$	RND	TYP
ASC	SUB$	SIN	SGN	REC
HEX$	INSTR	COS	ABS	

Program Resident Data Statements:
DATA
READ
RESTORE

Machine Statements and Functions:
INVOKE
PERFORM
EXFN
EXFN%

File I/O:
CATALOG
DELETE
RENAME
LOCK
UNLOCK
CREATE
OPEN # filenumber (AS INPUT, AS OUTPUT, AS EXTENSION)
CLOSE # filenumber
CLOSE
INPUT # filenumber, recordnumber
OUTPUT # filenumber
PRINT # filenumber, recordnumber
PRINT # filenumber, recordnumber USING
READ # filenumber, recordnumber
WRITE # filenumber, recordnumber

Console I/O:
INPUT
GET
TAB
SPC
SCALE
PRINT
PRINT USING
IMAGE specification(s)

String specifications:
A reserves a character position left-justified
C reserves a character position center-justified
R reserves a character position right-justified

Literal specifications:
X prints a space
/ prints a carriage return/line feed
"literal" prints whatever is in quotes

Digit specifications:
reserves one numeric digit, leading zeros suppressed
Z reserves one numeric digit, leading zeros printed
& reserves one numeric digit or comma (comma fill every three digits)
. reserves a position for the decimal point
+ reserves a position for the sign
− reserves a position for the sign (if negative)
$ reserves a position for the dollar sign
** asterisk fill
+ + floating sign
− − floating sign (if negative)
$$ floating dollar sign

Scientific notation specification:
E reserves a position for the exponent (power of 10)

Engineering notation specification:
same as scientific notation, except the exponent is always a multiple of three

Apple Computer Inc. 10260 Bandley Drive Cupertino, California 95014 (408) 996-1010 TLX 171-576
©1981 Apple Computer Inc.
Printed in U.S.A. 11/81 250M Order No. A2F0036

apple computer

Apple *III* Pascal

A Powerful Language and Development Environment

Languages and Utilities

Apple III Pascal is a fully integrated software development environment. It incorporates an Apple III adaptation of the UCSD Pascal™ Operating System with the Apple III's Sophisticated Operating System (SOS). Because it offers a compiled language, the Apple III Pascal system enables programs to execute quickly in minimal space.

The discipline imposed by structured programming facilities and extensive data structures helps programmers work more efficiently, thus making Apple III Pascal the language of choice for large business, scientific, and educational programs.

Benefits

Apple *III* Pascal...

- *increases programmer productivity because it provides a total software development facility...*
- *simplifies program design through Pascal's structured programming environment and rich variety of data types...*
- *lowers software development costs because it is a powerful, high-level language...*
- *is source-program compatible with routines written in Apple II Pascal, so programs can be easily transported without being redeveloped...*
- *cuts maintenance costs because Pascal's modular design helps the programmer write well-structured, easy-to-understand programs...*
- *expands your programming workspace, because it enables adaptation to available memory...*
- *optimizes your use of available memory, because its command processor allows each application to use only as much graphics space as needed...*
- *includes a powerful, easy-to-use text editor that affords documentation and source code maintenance...*
- *provides flexible input/output through its user-transparent access to Apple III's SOS.*

Apple *III* Pascal— A Closer Look

Apple *III* Pascal meets the programming requirements of a wide range of users. Its variety of data types, control functions, and powerful utility routines can dramatically improve a programmer's productivity on large projects. This is one reason why Apple Computer has selected Pascal as a standard vehicle for most system and applications development.

Why UCSD Pascal?

Apple *III* Pascal provides the programmer with a high level of portability because it's based on UCSD Pascal, the microcomputer industry standard. Users can access large existing libraries of Pascal programs, and can run Apple *III* Pascal programs with minimum conversion on most computer systems offering Pascal.

UCSD Pascal is more than a language. Compiler, Assembler, Editor, Linker, and Filer are all integrated within a single, powerful system. Built-in procedures and functions enhance the string, byte, and I/O capabilities. This total package provides the user with a comprehensive set of software tools for program development, and the ability to create and maintain program libraries.

Why Pascal For Apple *III*?

Apple has taken advantage of all of UCSD Pascal's features while offering additional capabilities. They include:

- up to 64K bytes of memory for data;
- code space limited only by total memory size of the machine;
- conformance to the IEEE Proposed Standard for 32-bit Binary

Floating-Point Arithmetic (this standard specifies special optional treatment of exceptions like overflow, configurable arithmetic under program control, and accuracy of arithmetic, I/O conversions, remainder, and square root);
- easy access to Apple ///'s human interface features—color graphics, speaker, joysticks, and keyboard;
- transcendental functions;
- extensive documentation.

System Configuration

To use Apple /// Pascal, you will need the following system:
- an Apple /// system with 128K bytes RAM;
- a suitable high-resolution video display device (Monitor /// recommended);
- one Disk /// expansion drive.

Technical Specifications

Format:
16-sector diskettes, 140K (143,360) bytes per diskette.

Editor:
Editing functions include cursor control, text modification, formatting, searching and marking capabilities. The Editor offers:
- fast, screen-oriented editing for program development and text editing;
- 80-character lines;
- upper and lower case;
- editing capabilities for Apple Business BASIC ASCII data files.

Compiler:
- Translates Pascal source text into P-code.
- Source language procedures or functions can be compiled separately as UNITs, allowing a large program to be subdivided into smaller, more manageable parts. Commonly used procedures and functions can be compiled once, stored in a library file, and accessed by other programs through the Linker.
- EXTERNAL routines can be declared and later linked into the host program by the Linker.
- Compiler option specifications can be embedded in the source text in order to control listings, suppress screen messages, generate range-checking code, allow GOTO statements, or include separate source files.

Assembler:
The Apple /// Pascal Assembler converts assembly language routines into code files that can be linked with a Pascal host program. The Apple /// Pascal Assembler is a version of the UCSD Adaptable Assembler, implemented specifically for the 6502 microprocessor used in the Apple /// computer.

Linker:
The Apple /// Pascal Linker lets you combine code files (which may be compiled P-code or assembled machine code) into a specified code file. This lets you incorporate useful routines into programs without having to rewrite, recompile, or reassemble these routines.

Filer:
The Filer handles chores, such as storing and retrieving data on disk, moving and deleting disk files, and creating or modifying diskette directories. It also:
- provides complete control over all SOS file facilities, including directories, sub-directories, and data transfer;
- sets or changes prefix so that long filenames can be entered easily;
- manages the assignment of system workfiles.

System Library:
The Apple /// Pascal system includes a utility program that will collect separately compiled or assembled routines into a library file. The System Library contains:
- The PGRAF intrinsic unit, which provides easy access to Apple ///'s high-resolution color capability. PGRAF provides a set of procedures for plotting dots and lines, setting colors, and defining the viewport. A

feature of the display is an invisible cursor which is used as a position reference in certain operations. Procedures exist for moving the cursor without affecting the display.
- The APPLESTUFF intrinsic unit, which contains routines to generate random numbers, interface with joysticks, and generate sound on the Apple ///'s speaker.
- The TRANSCEND intrinsic unit, which contains transcendental functions useful for mathematical calculations.
- The CHAINSTUFF intrinsic unit, which allows one program to "chain to" another program.

The Apple /// Pascal Package Order No. A3D0005

With your Apple /// Pascal order, you will receive:
- three Apple /// Pascal System diskettes;
- one blank diskette;
- Apple /// Pascal Introduction, Filer, and Editor manual;
- Apple /// Pascal Program Preparation Tools manual;
- Apple /// Pascal Programmer's Manual (Volumes I and II).

Apple Computer Inc. 10260 Bandley Drive Cupertino, California 95014 (408) 996-1010 TLX 171-576
©1981 Apple Computer Inc.
Printed in U.S.A. 11/81 250M Order No. A2F0035

apple computer

Apple /// Record Processing Services (RPS)

A Sophisticated Multi-Keyed Access Method for Pascal Programmers

Apple's Record Processing Services offers software developers a sophisticated multi-keyed file access standard for the Apple III.

Designed for both flexibility and high performance, Apple III RPS provides a solid base for application programs that manage large quantities of information. It handles data storage and retrieval at the record level, permitting both random and indexed sequential access to files. It also creates, deletes and erases files.

Apple III RPS, a Pascal intrinsic unit requiring no linking, uses an enhanced B+ tree structure and is fully integrated with the Apple III SOS (Sophisticated Operating System). It supports the ProFile 5-megabyte hard disk or any block device supported by SOS.

Benefits

Apple /// RPS...

- *provides compatibility with other RPS-based Apple III programs and saves costly development time...*
- *increases programming flexibility with a complete range of 11 data types, six access modes, and two record types...*
- *provides sophisticated capabilities such as concurrency control, multifield keys, and multiple simultaneous file accesses on one file...*
- *simplifies programming through the use of data structures for passing all parameters...*
- *permits the development of serious application programs that can store up to 16 megabytes of data...*
- *optimizes performance by allowing you to configure memory usage.*

Apple /// RPS— A Closer Look

Random and Sequential Access

Apple /// RPS permits the programmer to store and retrieve data by key values rather than by a specific physical disk address. Each data record has associated with it from one to eight keys that may be used for accessing the record.

Apple /// RPS provides two types of access to your data records: random and sequential. You can access records randomly by specifying a particular key value or sequentially through the order associated with one of the keys. Apple /// RPS supports both forward and backward sequential access through your file.

Enhanced B+ Tree Index Structure

Apple /// RPS employs the sophisticated B+ tree index structure. A B+ tree provides the following advantages:

- uniform performance as the file size grows;
- rapid retrieval, insertions and deletions of data;
- linked key values at the lowest level for rapid sequential access through the file;
- automatic reuse of space generated from deleted key values;
- no need for reorganization of files;
- retrieval of records accomplished without a prior sort.

Program Calls and Access Structures

Apple /// RPS is written in Pascal. The interface to Pascal consists of two components: program calls and data (access) structures. The program calls tell RPS which actions to perform, and the access structures are used to exchange information about that action. All RPS calls use an access structure for communicating information and passing parameters.

Creating and Accessing Files

To use Apple /// RPS, the programmer must first create an empty RPS file. An empty file consists of definitions of record size and descriptions of keys. RPS creates two SOS files for every RPS file you open—a data file and an index file—but their existence is transparent to the programmer.

Once the RPS file exists, the user may access it (called "opening a scan" on the file) to insert, locate, and manipulate individual records. A scan uses one and only one index, which determines the order of the records for that scan. An internal pointer, the Current Record of Scan (CROS), keeps track of the programmer's place in the file for each scan.

Access Modes

The random and sequential record access provided by Apple /// RPS consists of six access modes: Direct, Current, First, Last, Next, and Prior. Access modes determine the way in which records are retrieved. The programmer can switch from mode to mode during a file scan simply by selecting a new access mode.

Sparse Files

Apple /// RPS exploits the sparse file capability provided by SOS. Although it logically reserves 16 megabytes for each RPS file you open, it only allocates space on disk as the data are actually written out.

Support Calls

Apple /// RPS provides important support calls for the programmer's convenience. These include the ability to control storage allocation and to return information about files and descriptions of keys.

Key Matches

Apple /// RPS supports exact key matches and can approximate partial key matches by using multiple RPS calls. RPS does not support generic (wildcard) matches.

User-Defined Sort Order

If the data types supported by RPS are not sufficient, the user may define his or her own data type. In order for RPS to support this data type, the user must supply two assembly routines. One routine compares values of this data type, and the other determines null values. Once these routines are loaded into RPS, RPS will then be able to sort the key values of the unknown data type.

System Configuration

To use Apple /// RPS, you need the following system:

- an Apple /// system with at least 128K bytes RAM;
- Monitor /// or other suitable video display device;
- one external drive (either Apple Disk /// or ProFile hard disk drive);
- Apple /// Pascal (A3D0005).

Technical Specifications

Program Calls:

File Level Calls

RPS__CREATE (FASpointer)	Creates an empty RPS file.
RPS__DESTROY (FASpointer)	Deletes an RPS file in its entirety.
RPS__ERASE (FASpointer)	Deletes all data from an RPS file, leaving an empty file.
RPS__FILEINFO (FISpointer)	Returns information about a specific file.
RPS__KEYINFO (KISpointer)	Returns information about a specific key.
RPS__SEGINFO (SISpointer)	Returns information about a specific segment.

Record Level Calls

RPS__OPENSCAN (SASpointer)	Initiates a scan on an RPS file
RPS__CLOSESCAN (SASpointer)	Terminates a scan on an RPS file.
RPS__FETCH (SASpointer)	Retrieves a record from an RPS file.
RPS__POSITION (SASpointer)	Retrieves a key value from a key index.

RPS__INSERT (SASpointer)	Adds a record to an RPS file.
RPS__UPDATE (SASpointer)	Rewrites a record in an RPS file.
RPS__DELETE (SASpointer)	Removes a record from an RPS file.

Other Support Calls

RPS__PTR__LOAD (varname)	Returns a pointer to a variable.
RPS__INITIALIZE (ISpointer)	An optional call that computes and allocates space for control structures and buffers for use by RPS.

Access Structures:

File Level Access Structures		Components
FAS	File Access Structure	pointer to filename; record size; record type; page size; pointer to KD; status
KD	Key Descriptor	duplicates allowed; nulls allowed; page size; pointer to next KD; pointer to SD
SD	Segment Descriptor	segment position, size, data type; sort order (ascending or descending); pointer to next SD
FIS	File Information Structure	pointer to filename; number of indexes; page size; record size and count; status
KIS	Key Information Structure	pointer to filename; key identifier; index page size; duplicates allowed; nulls allowed; number of segments, key values, levels in index; status
SIS	Segment Information Structure	pointer to filename; key and segment identifier; positions in key and record; segment size; data type; sort order; status

Record Level Access Structure		Components
SAS	Scan Access Structure	scan identifier; pointer to filename; processing allowed (read only or modify); access mode; key identifier; key area pointer; record area pointer; record locking; file locking; status

Support Access Structure		Components
IS	Initialization Structure	key size; number of keys, segments, files and scans; number of buffers of 512, 1024, 1536 and 2048 bytes; extend size; release priority; write threshold; status

Access Modes:

FIRST	Positions the CROS between start-of-file and the first record.
LAST	Positions the CROS between the last record and end-of-file.
CURRENT	Selects the record to which the CROS currently points.

DIRECT	Positions the CROS at the record whose key value the user supplies.
NEXT	Positions the CROS at the record following the record to which the CROS currently points. Order is determined by the key index being used.
PRIOR	Positions the CROS at the record preceding the record to which the CROS currently points. Order is determined as in NEXT.

The record level calls POSITION and FETCH support all six access modes, whereas INSERT, UPDATE and DELETE support only CURRENT.

Keys:

- Number of keys: minimum of one, maximum of eight.
- Number of segments (fields or parts of fields) per key: minimum of one, maximum of eight.
- Segments do not have to be physically contiguous or in any particular place in the record.
- The first field of a record does *not* need to be part or all of a key.
- Maximum key size: 504 bytes.
- Key values may be null or allow duplicates. Once key is defined as null and/or duplicate values allowed, definition may not be changed.
- Two ways to order each key segment: ascending or descending.

Records:

- Maximum record size: 2K bytes; fixed or variable length records with fixed length keys.
- Number of records per file: depends on record size; maximum of 16M bytes of data and 2M bytes of key values per index.
- Optional concurrency control.
- Automatic reuse of space generated from deleted records.

Files:

- Number of files open simultaneously: constrained by memory.
- Number of scans (file accesses) open simultaneously: constrained by memory; no architectural limit. Scans may be on the same index or different indexes.
- File locking: shared or exclusive use.

Data Types:

- Data types supported by RPS: string; unknown (user-defined sort order);
16 bit integer: signed or unsigned;
32 bit integer: signed or unsigned; 32 bit real;
64 bit integer: signed or unsigned;
packed array of byte; packed array of character.

License Information

If you are developing software with Apple /// RPS, you will need to obtain a low cost license agreement from Apple Computer, Inc. This agreement protects Apple's copyrights and trademarks. Technical support is available to licensed developers.

- Limited Distribution Computer Software License Agreement

For developing software with Apple /// RPS that is either for resale or distribution.

- Multi-Use License Agreement

For shared access to Apple /// RPS through a hard disk network.

For more information, please contact this office:
Vendor Support/PCSD Technical Support Department
Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, CA 95014

The Apple /// RPS Package Order No. A3D0018

With your Apple /// Record Processing Services order, you will receive the following items:

- Apple /// RPS diskette;
- RPS Programming Manual.

The specifications of products may change without notice.

Apple Computer Inc. 20525 Mariani Avenue Cupertino, California 95014 (408) 996-1010 TLX 171-576

©1982 Apple Computer Inc. Printed in U.S.A. 5/82 200M Order No. A3F0072